VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Ondřej Zelený

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV RADIOELEKTRONIKY
DEPARTMENT OF RADIOENGINEERING

PROGRAM PRO DEMONSTRACI LINKOVÝCH KÓDŮ
PROGRAMME FOR LINE CODE DEMONSTRATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Ondřej Zelený**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. Aleš Prokeš, Ph.D.**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Elektronika a komunikační technologie**

Ústav radioelektroniky

Student: Ondřej Zelený ***ID:*** 195654
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Program pro demonstraci linkových kódů

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s vlastnostmi nejběžnějších linkových kódů a dále s metodami programování generátorů libovolných průběhů (tzv. arbitrary waveform generator) LeCroy LW410, Siglent SDG2066X nebo podobných pomocí PC. Vytvořte sadu funkcí v Matlabu pro přenos dat do generátoru, generování linkových kódů a nastavování základních parametrů generovaného signálu (amplituda, kmitočet) a ověřte jejich správnou funkci.

Pomocí grafického rozhraní (GUI) vytvořte vhodné uživatelské prostředí pro snadné ovládání generátoru. Je požadována možnost generování linkových kódů na základě bitové posloupnosti zadané v okně aplikace, náhodně generované nebo uložené v souboru. Dále je požadována možnost nastavení typu kódu, jeho parametrů a zobrazení časového průběhu a spektra. Vytvořte návod pro laboratorní měření. K dispozici bude již vypracovaný podobný projekt pro starší verzi Matlabu, který není s posledními verzemi Matlabu plně kompatibilní, avšak který lze použít jako předlohu.

DOPORUČENÁ LITERATURA:

[1] PROKEŠ, A. Rádiové komunikační systémy. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. 2013.

[2] LeCroy WaveStation LW4001LW400A Series AWG Remote Programmer's Manual [online]. Chestnut Ridge: LeCroy Corporation 1996 - [cit. 20. května 2010]. Dostupné na www: http://www.lecroy.com/tm/library/manuals/download.asp?id=800.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 4.6.2020

Vedoucí práce: prof. Ing. Aleš Prokeš, Ph.D.

prof. Ing. Tomáš Kratochvíl, Ph.D.
předseda rady studijního programu

Abstrakt

Tato bakalářská práce má za cíl návrh programu pro demonstraci linkových kódů v prostředí Matlab. Teoretická část se zabývá teoretickými podklady, které jsou následně využity v praktické části k tvorbě programu.

Klíčová slova

Linkové kódy, SCPI, Matlab, Generátor libovolných signálů

Abstract

The goal of this term paper is design program for demonstation of line encoding in Matlab. Theoretical part explains theory needed for implementation. Practical part then uses theory from previous part to implement and design application in Matlab.

Keywords

Line codes, SCPI, Matlab, Arbitrary waveform generator

Bibliografická citace:

ZELENÝ, Ondřej. Program pro demonstraci linkových kódů [online]. Brno, 2020 [cit. 2020-05-28]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/126106. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky. Vedoucí práce Aleš Prokeš.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Program pro demonstraci linkových kódů jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **4. června 2020** ……

podpis autora

Poděkování (nepovinné)

Děkuji vedoucímu bakalářské práce prof. Ing. Aleši Prokeši, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **4. června 2020** ……

podpis autora

Obsah

Seznam symbolů a zkratek

Zkratky:

FEKT	…	Fakulta elektrotechniky a komunikačních technologií
VUT	…	Vysoké učení technické v Brně
NRZ	…	Non return to zero
RZ	…	Return to zero
AMI	…	Alternate mark inversion
SCPI	…	Standard Commands for Programable Instruments

Symboly:

T_b	…	Bitová perioda	[s]
tTb	…	Počet vzorků na periodu	[-]
A	…	Napěťová úroveň	[V]
N	…	Počet bitů	[-]
ch	…	Kanál	[-]
com	…	COM port	[-]

Seznam obrázků

Seznam tabulek

Seznam rovnic

ÚVOD

Cílem práce je vytvoření programu pro demonstraci nejběžnějších linkových kódů a následný přenos těchto signálů do generátoru libovolných průběhů. Program bude schopen generovat 15 linkových kódu včetně jejich vizualizace a výpočtu jejich spekter. Dále bude schopen přenášet vytvořené signály do generátoru pomoc USB rozhraní a nastavovat jeho základní parametry.

Práce je rozdělena do tří kapitol, počínaje kapitolou **Teoretický úvod**, kde jsou vysvětleny všechny potřebné informace využité v kapitole dva, **Praktická část**, kde jsou tyto informace využity k samotné implementaci a tvorbě funkcí a grafického rozhraní. Nakonec je zde kapitola **Závěr**, kde jsou shrnuty dosažené výsledky a popsané problémy, se kterými jsem se v průběhu implementace funkcí setkal.

1. TEORETICKÝ ROZBOR

1.1 Linkové kódy

Linkové kódování, které tvoří způsob přenosu dat v základním pásmu, našlo velké využití především v komunikačních technologiich. Obecně se jedná o způsob kódování digitálního signálu určených k přenosu v základním pásmu. Do dnešní doby bylo vyvinuto velké množství linkových kódů pro přizpůsobení různým charakteristikám kanálů, bitové rychlosti, technologii a různým výkonovým požadavkům. Vybráním vhodného linkového kódu tak lze ovlivnit šířku pásma přenášeného signálu, jeho stejnosměrnou složku, a dokonce tak i lze zajistit synchronizaci přijímače s vysílačem. Linkové kódy lze rozdělit podle tří základních kritérií: [6][7][8]

- Podle počtu úrovní
  - Dvojúrovňové
  - Trojúrovňové
  - Víceúrovňové

- Podle návratu k nule
  - S návratem k nule (Return to Zero - **RZ**)
  - Bez návratu k nule (Non Return to Zero - **NRZ**)

- Podle napěťových úrovní signálu
  - Unipolární (signálové prvky mají pouze kladnou nebo zápornou polaritu)
  - Bipolární (signálové prvky nabývají odlišných polarit)

1.2 Typy linkových kódů

1.2.1 Non-Return to Zero

NRZ je jednoduchý linkový kód s dvěma stavy bez návratu k nule. Každá logická úroveň je reprezentována jinou napěťovou úrovní. Nevýhoda tohoto linkového kódu spočívá v přítomnosti stejnosměrné složky, která může způsobovat problémy při přenosu. Dalším problém NRZ kódu nastává při dlouhé sekvenci stejné úrovně, která stěžuje synchronizaci mezi vysílačem a přijímačem. [5][9][10]

1.2.1.1 NRZ-L

Nejjednodušší varianta NRZ -L (Level), kde logická 1 je vyjádřena vysokou úrovní a logická 0 nízkou úrovní. [11]

1.2.1.2 NRZ-M

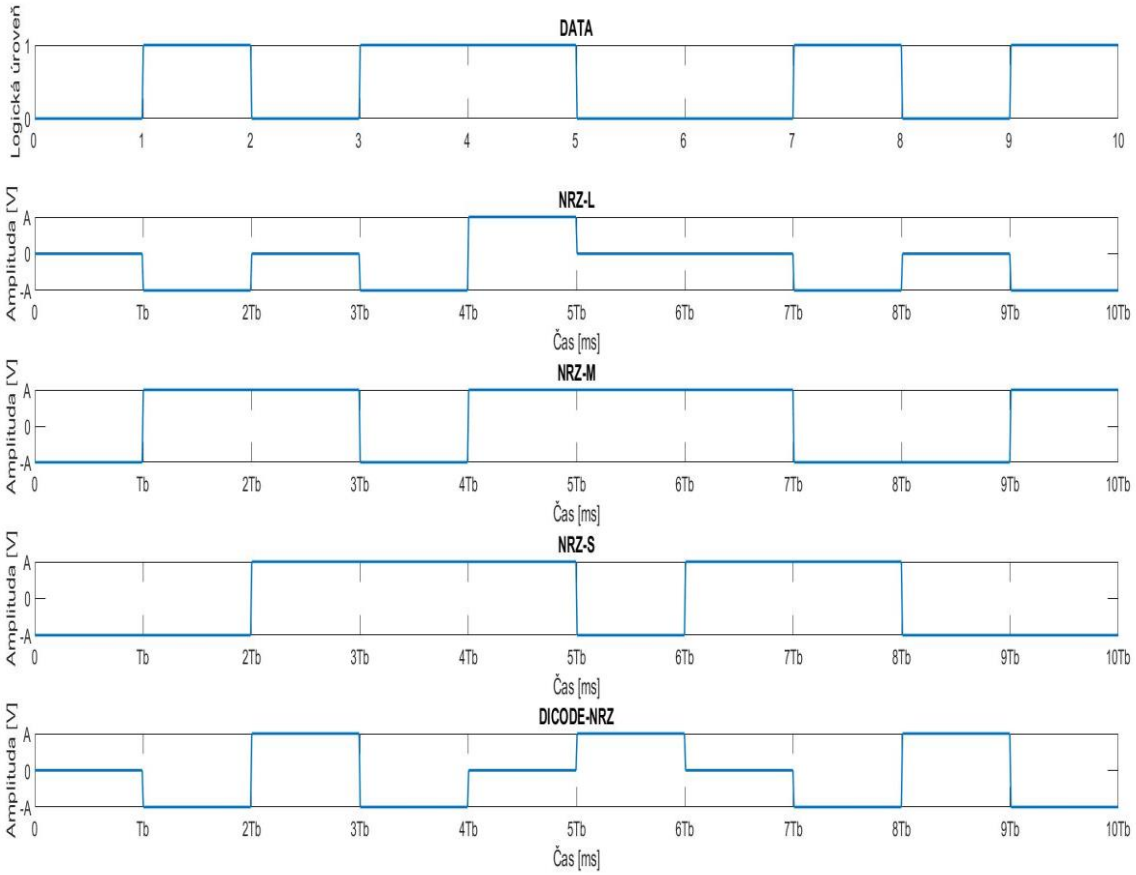
Varianta NRZ-M (Mark), kde je logická 1 je vyjádřena přechodem mezi vysokou a nízkou úrovní a naopak a logická 0 zachovává úroveň předchozího bitu. [11]

1.2.1.3 NRZ-S

U NRZ-S (Space) je naopak logická 1 vyjádřena zachováním úrovně předchozího bitu a logická 0 je vyjádřena přechodem mezi vysokou a nízkou úrovní a naopak. [11]

1.2.1.4 DICODE NRZ

Tříúrovňová varianta, kde přechod z logické 0 do logické 1 je symbolizován kladným pulzem o šířce T_b a přechod z logické 1 do logické 0 záporným pulzem délce T_b. Přechody z logické 0 do logické 0 a logické 1 do logické 1 mají nulovou úroveň. [11]



Obr. 1 Non-Return to Zero

1.2.2 Return to Zero

Jak již z názvu patrně vyplívá, RZ kodovaný signál se v určitých úsecích vrací do napěťové nuly. Tento přechod nastává vždy v druhé polovině periody T_b. Nevýhodou tohoto linkového kódování je větší šířka pásma a také v případě unipolární či dicode varianty komplexnost signálu. [11]

1.2.2.1 Unipolarní RZ

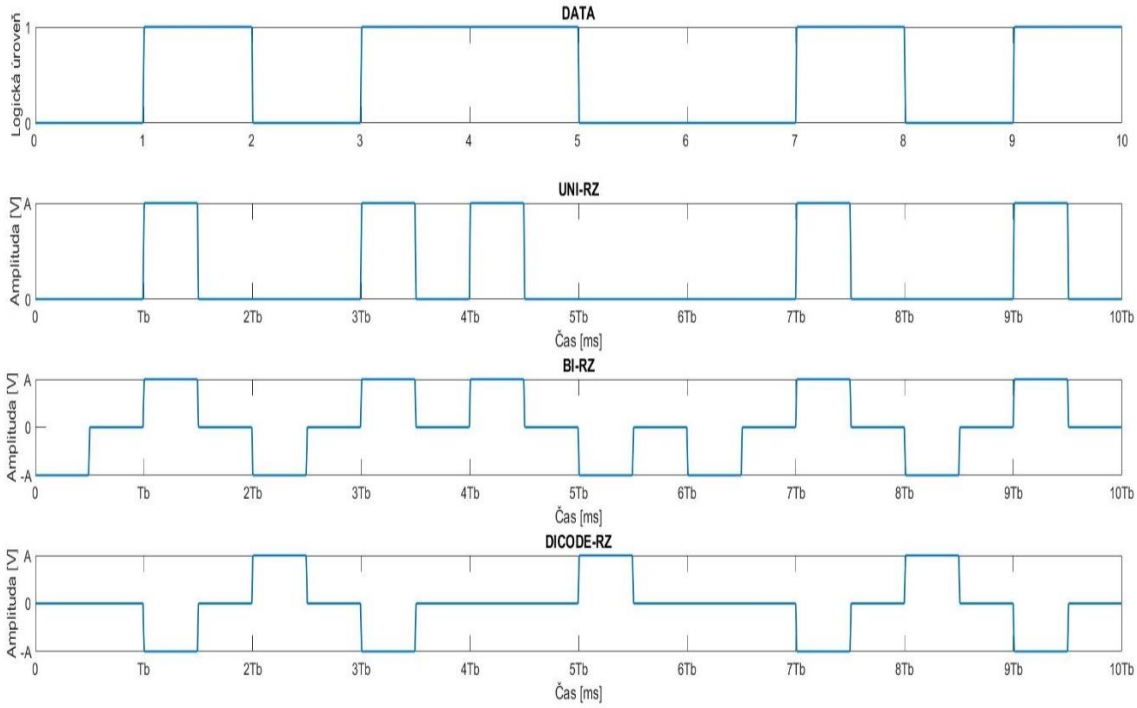
V tomto případě je logická 1 vyjádřena pulzem do vysoké úrovně, příčemž ve vysoké úrovni zůstává pouze polovinu bitové periody T_b a v druhé polovině má nulovou úroveň. Logická 0 je vyjádřena nulovou úrovní. [11]

1.2.2.2 Bipolarní RZ

Logická 1 je vyjádřena pulzem do vysoké úrovně o délce poloviny T_b, poté se vrací do nízké úrovně. Logická 0 je vyjádřena nízkou úrovní. [11]

1.2.2.3 Dicode RZ

Tříúrovňová varianta, kde přechod z logické 0 do logické 1 je symbolizován kladným pulzem o šířce $T_b/2$ a v druhé polovině se vrací do nulové úrovně a přechod z logické 1 do logické 0 záporným pulzem délce $T_b/2$ a v druhé polovině se opět vrací do nuly. Přechody z logické 0 do logické 0 a logické 1 do logické 1 mají nulovou úroveň. [11]



Obr. 2 Return to Zero

1.2.3 Bi-Phi

1.2.3.1 Bi-Phi-L

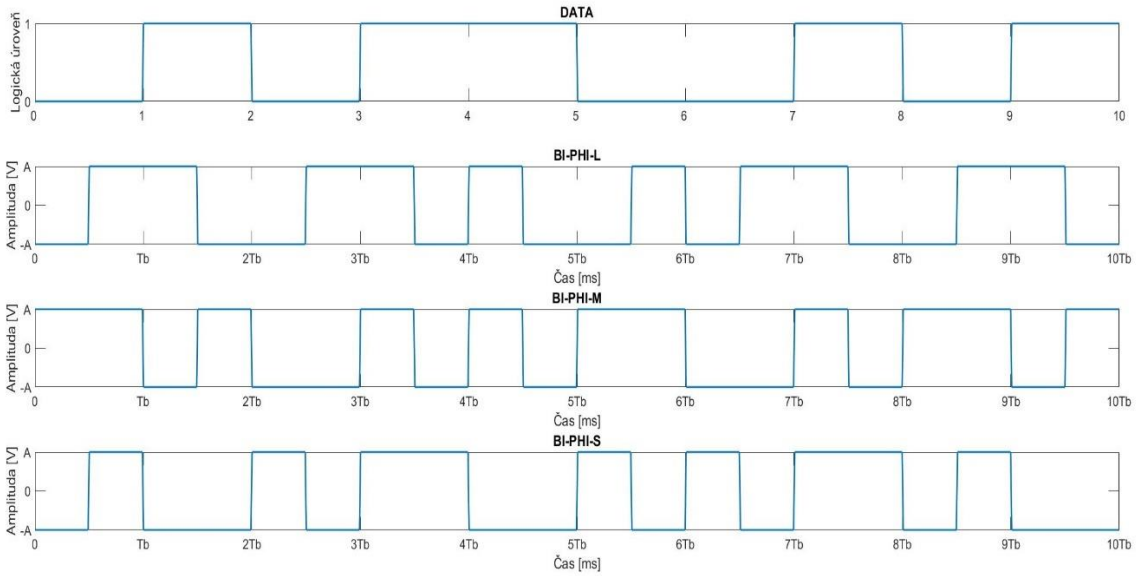
Bi-PHI-L (Level) linkový kód využívá obě poloviny bitové periody a to tak, že pro logickou 1 je kladný impulz v první části a v druhé části má nulovou úroveň, naopak pro logickou 0 je první poloviny bitové periody nulová a v druhé polovině je kladný impulz.[10] [11]

1.2.3.2 Bi-Phi-M

Tato varianta využívá pro logickou 1 pulz v první i ve druhé polovině bitové periody a to na základě toho, jakou měl signál v předchozí periodě úroveň. Pokud logická 1 následuje po nízké úrovni, je impulz o šířce $T_b/2$ v první části periody a v druhé má nulovou úroveň. Následuje-li logická 1 po vysoké úrovni, je první půlperioda nulová a v druhé půlperiodě je impulz do vysoké úrovně. Obdobné pravidlo platí pro logickou 0, kde má celá bitová perioda jednu úroveň a úroveň je vždy opačná než je úroveň na konci předchozí periody. [10] [11]

1.2.3.3 Bi-Phi-S

Pravidla pro logické hodnoty jsou opačná než u Bi-Phi-M. Pro logickou 0 je pulz v první i ve druhé polovině bitové periody a to na základě toho, jakou měl signál v předchozí periodě úroveň. Pokud logická 0 následuje po nízké úrovni, je impulz o šířce *Tb*/2 v první části periody a v druhé má nulovou úroveň. Následuje-li logická 0 po vysoké úrovni, je první půlperioda nulová a v druhé půlperiodě je impulz do vysoké úrovně. Pravidlo pro logickou 1, kde má celá bitová perioda jednu úroveň je takové, že úroveň je vždy opačná než je úroveň na konci předchozí periody. [10] [11]



Obr. 3 Bi Phase

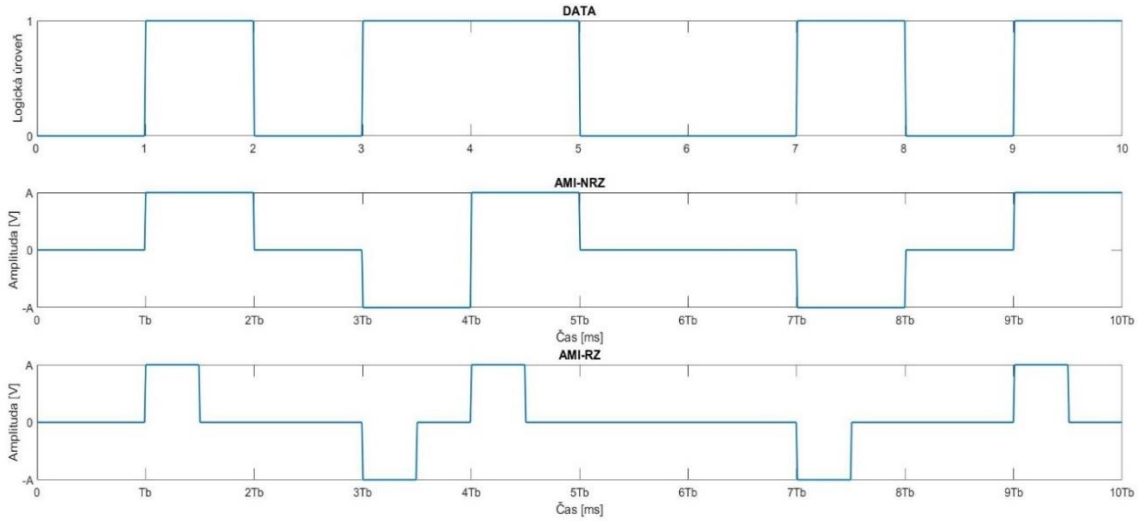
1.2.4 AMI

1.2.4.1 NRZ-AMI

Jedná se o tří-stavový linkový kód, s alternující úrovní jedničky, to znamená že jednička je vyjádřena jako kladný i záporný půlz s amplitudou A o délce T_b. Logická 0 je poté vyjádřena nulovým napětím o délce T_b. [11]

1.2.4.2 RZ-AMI

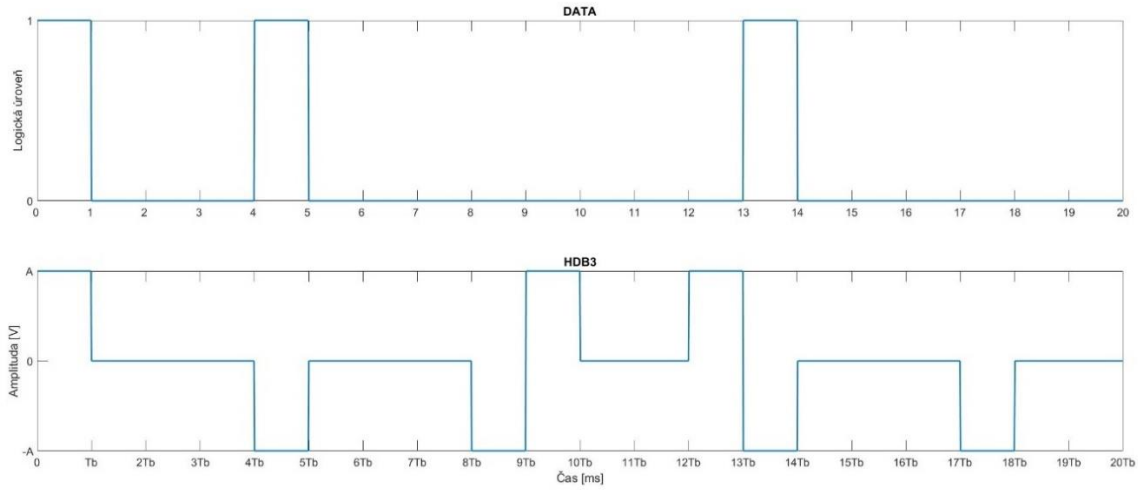
Tato varianta AMI kódu má opět alternující logickou 1, avšak jedná se o RZ, tzn. v polovině bitové periody se signál vrací do nuly. Logická 0 je kódována stejně jako u NRZ-AMI a to nulovou úrovni o délce T_b. [11]



Obr. 4 Alternate Mark Inversion

1.2.5 HDB3

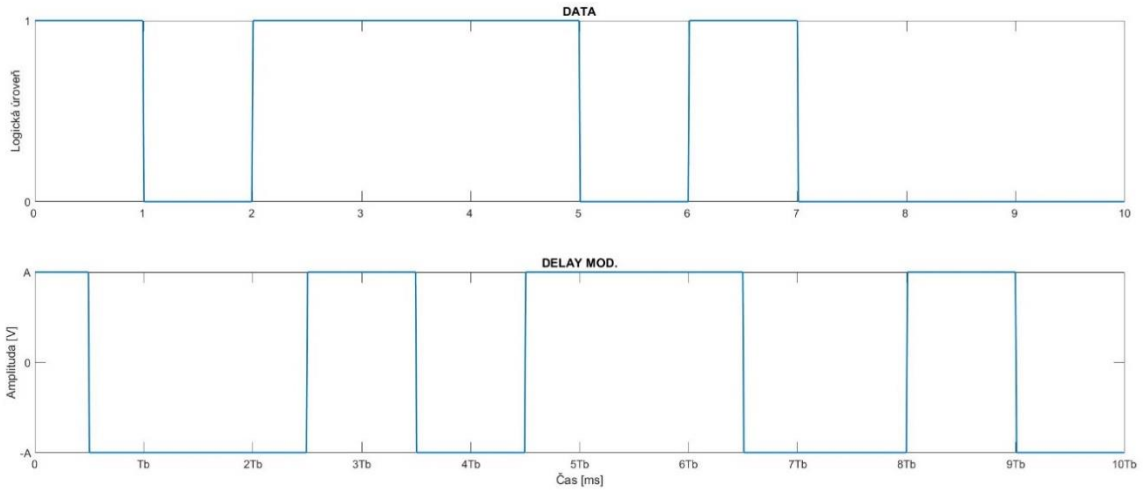
Jde o modifikovanou variantu linkového kódu AMI se specifickým kódováním pro sekvenci čtyř po sobě jdoucích nul. Pravidlo pro kódování logické 1 je totožné u AMI, tzn., střídavě kladný i záporný pulz o výšce A a délce T_b. Kódování logické 0 je buď jako nulová úroveň nebo pro sekvenci čtyř nul je závislé na počtu předchozích ať už kladných nebo záporných pulzů. Pokud je počet předchozích půlzů lichý, je sekvence nahrazena sekvencí ve tvaru 000X, kde X vyjadřuje pulz o výšce A a délce T_b s polaritou totožnou půlzu, který předchází sekvenci nul. Pokud je počet předchozích půlzů sudý, je sekvence nahrazena sekvenci ve tvaru X00X, kde X vyjadřuje pulz o výšce A a délce T_b s opačnou polaritou než je polarita pulzu, předcházejícího sekvenci nul.



Obr. 5 HDB3

1.2.6 Delay modulace

Tento druh kódování, využívá pulz o délce $T_b/2$ pro reprezentaci logické 1 a logickou nulu reprezentují periody, které mají po celou bitovou periodu stejnou úroveň. Logická 1 je tedy reprezentována pulzem o délce $T_b/2$ a střídavě se objevuje v první nebo v druhé polovině T_b, tak aby první perioda měla stejnou úroveň jako konec předcházející periody. Logická 0 mění úroveň pouze tehdy, následuje-li po logické 0 další logická 0, jinak zachovává předchozí úroveň.[7][11]



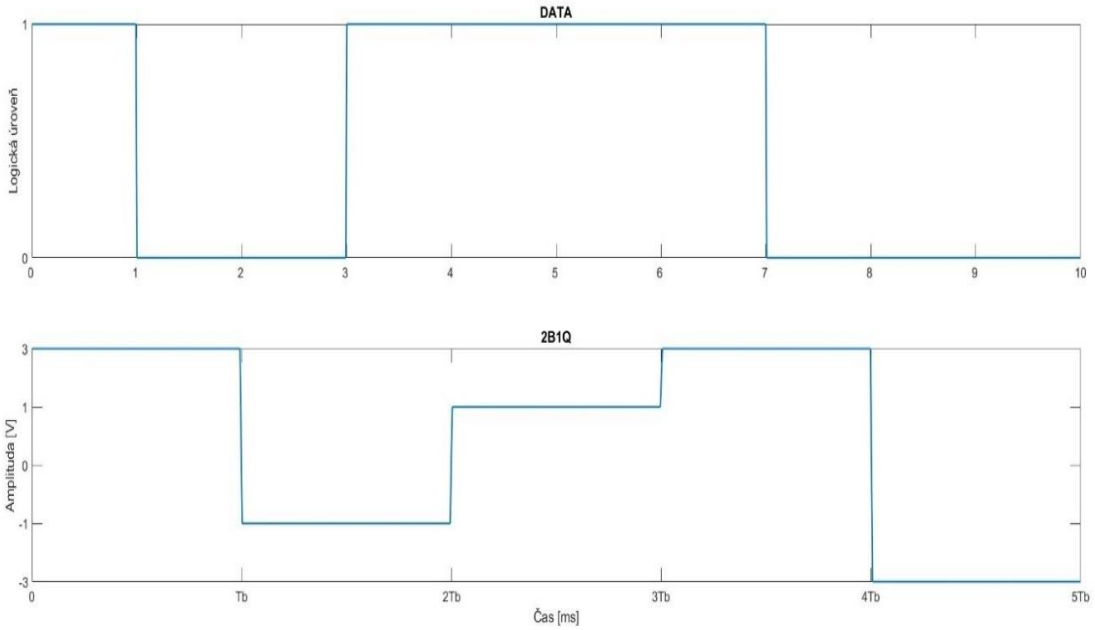
Obr. 6 Delay-modulation

1.2.7 2B1Q

Linkový kód 2B1Q, z anglického *2 Binary 1 Quaternary*, je čtyř-úrovňový linkový kód, každá reprezentující jinou kombinaci dvou bitů, podobně jak je tomu u QPSK modulace. Tímto způsobem kódování se dosáhne teoreticky až dvojnásobná rychlost přenosu, neboť se přenáší 2 bity v jedné periode T_b. Úrovně odpovídající binárním kombinacím jsou uvedeny v Tabulce 1.[4][5]

Tab. 1 Úrovně 2B1Q

Binární kombinace	2B1Q výstupní napětí [V]
00	-3
01	-1
10	+3
11	+1


DATA
Logická úroveň
0
1
0
1
2
3
4
5
6
7
8
9
10
2B1Q
Amplituda [V]
3
1
0
-1
-3
0
Tb
2Tb
3Tb
4Tb
5Tb
Čas [ms]

Obr. 7 2B1Q

1.3 SCPI

1.3.1 Úvod

SCPI, z anglického *Standard Commands for Programable Instruments,* popisuje strukturu příkazů a jejich syntaxi k ovládání programovatelných měřících přístrojů. Vývoj tohoto standardu započala společnost Hawlett Packard za účelem kompatibility přístrojů různých výrobců. Tento standard přinesl jednu obrovskou výhodu a to takovou, že se uživatel nemusí učit programovaní každého přístroje zvlášt, pouze se naučí standardní příkazy, kterým nyní rozumí téměř každý programovatelný přístroj.

1.3.2 Příkazy

Než se pustím do rozboru jednotlivých příkazu, je potřeba si o nich něco říci. Většinu SCPI příkazů lze zapsat krátkou, obvykle 3-4 písmenou formou nebo formou celého slova, které je zapsáno velkými písmeny a obvykle jednoslovně popisuje, co daný příkaz vykonává. Dalé lze příkazy rozdělit na **příkazy**(„Commands“) anebo **dotazy**(„Queries“). Queries se od příkazů liší v tom, že na konec jejich názvu(ať už zkratky nebo plného názvu) je doplněn symbol „**?**“, který už od pohledu mluví za vše. Všechny následující příkazy/dotazy a jejich odpovědi jsou popsány pro generátory řady SDG6000X značky Siglent.

1.3.2.1 *IDN

Jedná se o jeden ze základních **dotazů**, který požádá zařízení aby se identifikovalo.

Tab. 2 Syntaxe SCPI - *IDN

Syntaxe dotazu	*IDN?
Formát odpovědi	<Výrobce>,<model>,<seriové číslo>,<verze firmwaru>

1.3.2.2 *RST

Tento **příkaz**, iniciuje zařízení a nastaví jej do výchozího nastavení.

Tab. 3 Syntaxe SCPI - *RST

Syntaxe dotazu	*RST
Formát odpovědi	Bez odpovědi

1.3.2.3 WVDT

Tento příkaz získá nebo nastaví data libovolného průběhu. Dotaz lze zapsat ve dvou formátech. Formát 1 je pro získání dat vestavěných průběhů, kde *x* odpovídá indexu vestavěného průběhu(M0 – M196). Formát 2 je poté pro zskání dat uživatelem vytvořených průběhů.

Tab. 4 Syntaxe SCPI - WVDT

Syntaxe dotazu	Formát 1	WVDT? M*x*
	Formát 2	WVDT? USER,<wave_name>
Syntaxe příkazu	<Kanál>:WVDT <Index>,<Parametr>,<Hodnota>	
Formát odpovědi	Bez odpovědi	

Tab. 5 Syntaxe SCPI – WVDT popis

Parametr	Hodnota	Poznámka
WVNM	<název>	Název souboru v paměti AWG
LENGTH	<délka>	Délka v bytech [4B – 40MB]
FREQ	<frekvence>	Frekvence v Hertzích [Hz]
AMPL	<amplituda>	V_{pp}, ve voltech [V]
OFST	<offset>	Stejnosměrný offset ve voltech[V]
PHASE	<fáze>	Fázový posuv ve stupních [°]
WAVEDATA	<data>	

1.4 Signálové zpracování

1.4.1 Převzorkování

Převzorkování signálu s vzorkovacím kmitočtem $\mathbf{f_{vz}}$ na $\mathbf{Mf_{vz}/D}$, kde **M,D** jsou celá čísla. Signál je možné převzorkovat na vyšší ale také i na nižší vzorkovací kmitočet, v případě nižšího kmitočtu je nutné najít nejmenší společný násobek původního a nového vzorkovacího kmitočtu a z něj si poté odvodit faktory **M** a **D**. Obecný postup pro převzorkování obsahuje následující kroky[12]:

- **Interpolace** - Nejprve provedeme expanzi signálu na základě faktoru **M**. To probíhá tak, že se provede interpolace signálu vkládáním nulových hodnot mezi vzorky signálu. Poté se filtrací dolní propůstí s mezních kmitočetem na polovině původního vzorkovacího kmitočtu zajistí zachování pouze původních spektrálních složek. [12]
- **Decimace** – Následně probíhá redukce vzorku na základě faktoru **D**, to probíhá tak, že vybíráme pouze některé vzorky interpolovaného signálu. Ještě před tím je však nutné aplikovat antialiasingový filtr (opět dolní propust) s mezní frekvencí na polovině výsledného vzorkovacího kmitočtu. [12]

Vzhlem k tomu, že po interpolaci je využita dolní propust a před decimací je využita dolní propust, je zbytečné zařazovat do serie dvě dolní propusti, proto stačí zařazení pouze té, která má nižší mezní kmitočet. [12]

1.4.2 Spektrální analýza

K vypočtu spektra diskrétního(navzorkovaného) signálu je nutné využít diskrétní furierovu tranformaci (DFT). Pomocí DFT získáme komplexní koeficienty furierovy řady $\mathbf{X_k}$, které obsahují informaci o modulu a fázi. Tyto jednotlivé koeficienty popisují zastoupení konkrétní frekvence v signálu. Koeficienty lze vypočíst pomocí vzorce: [13]

$$\boldsymbol{X_k = \sum_{n=0}^{N-1} x_n * e^{-\frac{j2\pi kn}{N}}},$$ **[Rov. 1]**

kde **N** je počet vzorků signálu, $\mathbf{x_n}$ je n-tý vzorek a $\mathbf{X_k}$ odpovídá výkonu **k**-té frekvence v signálu. K výpočtu magnitudy a fáze slouží rovnice 2 a 3. [13]

$$\boldsymbol{Mod[X_k] = \sqrt{Re[X_k]^2 + Im[X_k]^2}}$$ **[Rov. 2]**

$$\boldsymbol{Faz[X_k] = tan^{-1}\left(\frac{Im[X_k]}{Re[X_k]}\right)}$$ **[Rov. 3]**

Spektrum se však obvykle neuvádí v magnitudě ale v dB nebo dBm. V našem případě je požadovan výkon v jednotkách dBm. K vypočtu výkonu v dBm lze využít rovnici 4.[13]

$$\boldsymbol{Výkon[dBm] = 10 * \log_{10}\left(\frac{P[W]}{1mW}\right)} \qquad \textbf{[Rov. 4]}$$

2. PRAKTICKÁ ČÁST

2.1 Funkce CreateWaveform

Tato funkce má 4 vstupní parametry,

- **Amplitude** (Amplituda generovaného signálu)[V]
- **Bit_period** (délka jednoho bitu)[ms]
- **Line_code** (zkratka linkového kódu) [-]
- **Sequence** (vstupní data ve formě vektoru) [-]

a 2 výstupní parametry

- **x**[vektor normalizovaného vystupního signálu]
- **T**[vektor časových stop pro každou amplitudu **x**]

Konstanty a proměnné zavedené v kódu.

tTb - počet vzorků na periodu
A - pomocná hodnota amplitudy (+-1)

Tato funkce vygeneruje normalizovaný vektor na základě vstupních parametrů, zobrazí průběh reálného signálu a vypočte jeho spektrum. Hlavní částí kódu této funkce je SWITCH, který na základě řetězce Line_code, spustí příslušnou část kódu. CASE tohoto SWITCHe rozeberu podrobněji v následujících podkapitolách.

2.1.1 NRZ-L

Na obrázku 8 je možná vidět implementace linkového kódu NRZ-L, kde v cyklu FOR(řádek 1.), čtu vstupní vektor dat (*Data(k)*) a na základě toho, jaká hodnota je na k-té pozici tohoto vektoru se rozhoduji příkazem IF-ELSE, zda se uplatní pravidlo pro logickou 1 (řádek 4), nebo pro logickou 0 (řádek 6)

```
1 case 'nrz-l'
2     for k = 1:N
3         if Data(k) == 1
4             x = [x ones(1,tTb)];
5         else
6             x = [x -ones(1,tTb)];
7         end
8     end
```

Obr. 8 Kód nrz-l

2.1.2 NRZ-M

Opět se jedná o relativně jednoduchou implementaci, každá generovaná jednička má opačnou polaritu než předchozí jednička. Proto je zde využita pomocná proměnná A, která změní polaritu po každé vygenerované sekvenci jedniček o délce *tTb* (řádek 4 a 5). Generuje-li se sekvence pro nulu, která zachovává předchozí polaritu, je proto sekvence jedniček zapisována s úrovní *-A* (nebot´ *A* změnilo polaritu během předcházející jedničky.

```
1 case 'nrz-m'
2     for k = 1:N
3         if Data(k) == 1
4             x = [x A*ones(1,tTb)];
5             A = -A;
6         else
7             x = [x -A*ones(1,tTb)];
8         end
9     end
```

Obr. 9 Kód nrz-m

2.1.3 NRZ-S

Zde je implementace opačná než u NRZ-M, jednička zachovává úroveň a nula alternuje mezi hodnotou *A* a *-A*. Tím pádem je zde kód téměž totožný, až na změněnou podmínku IF, která se nyní vykoná při logické nula a ELSE, který se vykoná při logické jedna.

```
1 case 'nrz-s'
2     for k = 1:N
3         if Data(k) == 0
4             x = [x A*ones(1,tTb)];
5             A = -A;
6         else
7             x = [x -A*ones(1,tTb)];
8         end
9     end
```

Obr. 10 Kód nrz-s

2.1.4 DICODE NRZ

V tomto případě už je implementace složitější, jelikož tento linkový kód je závislý na předchozí hodnotě. Z toho důvodu jsou tu dvě sady pravidel rozdělených příkazem IF-ELSE. Obsah IF (řádky 4 až 8) se vykonají pouze pro *k*=1,tzn. pouze v první smyčce cyklu FOR a rozhodne se zde o první bitové periodě signálu. Následně se vykonávají pouze pravidla obsažená v ELSE. Na řádcích 11-12 je pravidlo pro logickou nulu, která následuje po logické jedna (Kladný pulz o délce *Tb*/2). Naopak na řádcích 14-15 je pravidlo pro logickou jedna následující po logické nule. Pravidlo pro logickou 1 následující po logické 1 nebo logickou 0 následující po logické 0 je implementováno na řádcích 17-19.

```
1 case 'dicode-nrz'
2     for k = 1:N
3         if k == 1
4             if (Data(k) == 0)
5                 x = [x zeros(1,tTb)];
6             else
7                 x = [x -ones(1,tTb)];
8             end
9         else
10            if (Data(k) == 0) && (Data(k-1) == 1)
11                x = [x ones(1,tTb)];
12            end
13            if (Data(k) == 1) && (Data(k-1) == 0)
14                x = [x -ones(1,tTb)];
15            end
16             if ((Data(k) == 1) && (Data(k-1) == 1))
17                    || ((Data(k) == 0) && (Data(k-1) == 0))
18                x = [x zeros(1,tTb)];
19            end
20        end
21    end
```

Obr. 11 Kód dicode-nrz

2.1.5 UNI-RZ

U UNI-RZ se implementace zjednodušila pouze na několik řádku, neboť pro *Data(k)* rovno 1 se vykoná řádek 3 a do vektoru je přidáno *tTb*/2 jedniček a *tTb*/2 nul. Je-li *Data(k)* rovné 0 vykoná se řádek 6 a do výsledného vektoru je vloženo *tTb* nul.

```
1 case 'uni-rz'
2     for k = 1:N
3         if Data(k) == 1
4             x = [x ones(1,tTb/2) zeros(1,tTb/2)];
5         else
6             x = [x zeros(1,tTb)];
7         end
8     end
```

Obr. 12 Kód uni-rz

2.1.6 BI-RZ

Velice obdobná implementace jako u UNI-RZ s tím rozdílem, že logická nula je zde kódována jako záporný impulz o délce *tTb*/2 a poté se vrací do nuly. Logická jednička je pak kódována jako kladný impulz o délce *tTb*/2.

```
1 case 'bi-rz'
2     for k = 1:N
3         if Data(k) == 1
4             x = [x A*ones(1,tTb/2) zeros(1,tTb/2)];
5         else
6             x = [x -A*ones(1,tTb/2) zeros(1,tTb/2)];
7         end
8     end
```

Obr. 13 Kód bi-rz

2.1.7 DICODE RZ

Struktura je totožná jako u dicode nrz, pouze jsou zde zavedená pravidla pro RZ kód. V IF na řádku 3 se rozhoduje o první periodě, kde vycházím z předpokladu že předchozí úroveň byla nulová a vykoná se pouze v prvním cyklu. V následujících cyklech FOR se již vykonává pouze ELSE na řádku 10. Zde jsou opět zavedena pravidla pro 4 různé kombinace dvou po sobě jdoucích bitů.

```
 1     case 'dicode-rz'
 2         for k = 1:N
 3             if k == 1
 4                 if (Data(k) == 0)
 5                     x = [x zeros(1,tTb)];
 6                 end
 7                 if (Data(k) == 1)
 8                    x = [x -ones(1,tTb/2) zeros(1,tTb/2)];
 9                 end
10             else
11                 if (Data(k) == 0) && (Data(k-1) == 1)
12                     x = [x ones(1,tTb/2) zeros(1,tTb/2)];
13
14                 elseif (Data(k) == 1) && (Data(k-1) == 0)
15                     x = [x -ones(1,tTb/2) zeros(1,tTb/2)];
16
17                 elseif ((Data(k) == 1) && (Data(k-1) == 1))
18                         || ((Data(k) == 0) && (Data(k-1) == 0))
19                     x = [x zeros(1,tTb)];
20                 end
21             end
22         end
```

Obr. 14 Kód dicode-rz

2.1.8 BI-PHI-L

V tomto případě je implementace velice jednoduchá, kóduje-li se logická jednička, jedná se o přechod z vysoké úrovně do nízké a proto se do normovaného vektoru vkládá *tTb*/2 kladných jedniček a *tTb*/2 záporných jedniček (řádek 4). Pro logickou nulu je pravidlo opačné, přechod je z nízké úrovně do vysoké, tzn. do vektoru se vkládá *tTb*/2 záporných jedniček a *tTb*/2 kladných jedniček (řádek 6)

```
1 case 'bi-phi-l'
2     for k = 1:N
3         if Data(k) == 1
4             x = [x ones(1,tTb/2) -ones(1,tTb/2)];
5         else
6             x = [x -ones(1,tTb/2) ones(1,tTb/2)];
7         end
8     end
```

Obr. 15 Kód bi-phi-l

2.1.9 BI-PHI-M

Pro implementaci tohoto linkového kódu je opět využita pomocná proměnná *A*, která nám alternuje polaritu v tomto případě jedničky i nuly. Opět se zde při splnění podmínky na řádku 3 rozhodne o periodě na základě předpokladu že předchozí úroveň byla nízká. V dalších cyklech se již vykonává pouze ELSE na řádku 10. Vykoná-li se při *k*=1 pravidlo pro logickou jedničku (druhá polovina periody je v -1), polarita proměnné *A* se nezmění a ať už se vykoná v druhém cyklu pravidlo pro logickou jedničku nebo nulu, bude druhá perioda v případě logické jedničky mít kladný pulz o délce *Tb*/2 v první části bitové periody a v případě nuly bude kladný pulz o délce *Tb*. Vykoná-li se při *k*=1 pravidlo pro logickou nulu, je polarita *A* změněna a v druhém cyklu bude pro logickou jedničku generován pulz o délce *Tb*/2 v druhé části bitové periody a v případě logické nuly bude záporný pulz o délce *Tb*.

```
case 'bi-phi-m'
    for k = 1:N
        if k == 1
            if Data(k) == 1
                x = [x ones(1,tTb/2) -ones(1,tTb/2)];
            else
                x = [x ones(1,tTb)];
                A = -A;
            end
        else
            if Data(k) == 1
                x = [x A*ones(1,tTb/2) -A*ones(1,tTb/2)];
            else
                x = [x A*ones(1,tTb)];
                A = -A;
            end
        end
    end
```

Obr. 16 Kód bi-phi-m

2.1.10 BI-PHI-S

Totožný princip implementace jako u BI-PHI-M avšak se změněnými podmínkami. Zde je nula reprezentována pulzy o šířce *Tb*/2 v první nebo druhé polovině bitové periody a jednička je kódována jako pulz o šířce *Tb*. Tím pádem se jedná o totožný kód s jediným rozdílem, a to takovým, že nyní je pro IF podmínka *Data(k)* rovné 0.

```
case 'bi-phi-s'
    for k = 1:N
        if k == 1
            if Data(k) == 0
                x = [x -A*ones(1,tTb/2) A*ones(1,tTb/2)];
            else
                x = [x A*ones(1,tTb)];
            end
            A = -A;
        else
            if Data(k) == 0
                x = [x A*ones(1,tTb/2) -A*ones(1,tTb/2)];
            else
                x = [x A*ones(1,tTb)];
                A = -A;
            end
        end
    end
```

Obr. 17 Kód bi-phi-s

2.1.11 AMI-RZ

Velmi jednoduchá implementace, nula je jednoduše kódována nulovou úrovní (řádek 4) a jednička je kódována jako alternující pulz o délce *Tb*/2. Z toho důvodu je zde využita pomocná proměnná *A*, která změní polaritu po každé vygenerované sekvenci reprezentující logickou jedničku a díky tomu má následující sekvence reprezentující logickou jedničku opačnou polaritu než ta předchozí.

```
1 case 'ami-rz'
2     for k = 1:N
3         if Data(k) == 0
4             x = [x zeros(1,tTb)];
5         else
6             x = [x A*ones(1,tTb/2) zeros(1,tTb/2)];
7             A = -A;
8         end
9     end
```

Obr. 18 Kód ami-rz

2.1.12 AMI-NRZ

V tomto případě je opět nula reprezentována nulovou úrovní (řádek 4) a jednička je kódována jako alternující pulz o délce *Tb* (řádek 6). Po každé vygenerované sekvenci pro logickou jedničku (řádek 6) je změněna polarita pomocné proměnné *A*(řádek 7), takže následující sekvence pro logickou jedničku bude mít opačnou polaritu.

```
1 case 'ami-nrz'
2     for k = 1:N
3         if Data(k) == 0
4             x = [x zeros(1,tTb)];
5         else
6             x = [x A*ones(1,tTb)];
7             A = -A;
8         end
9     end
```

Obr. 19 Kód ami-nrz

2.1.13 HDB3

Jelikož je u tohoto kódu speciálně kódována sekvence nul, je zde zavedena další pomocná proměnná *ctnPulz*, která jak už napovídá název počítá počet vygenerovaných pulzů (kladných i záporných). Na rozdíl od předchozích kódů je zde využito cyklu WHILE, který umožní v závislosti na zpracovávaných datech měnit počet cyklů v průběhu některého z cyklů. Je-li *Data(k)* rovno jedné, je vygenerováno *tTb* kladných nebo záporných jedniček (řádek 6), změní se polarita pro zpracování další hodnoty a je incrementována pomocná proměnná *ctnPulz*. Pro *Data(k)* rovné nule se vykoná buď ELSEIF (řádek 10), kde se zkontrolují následující hodnoty vektoru Data a náchází-li se zde 4 po sobě jdoucí nuly je vygenerována sekvence ve tvaru 000X (X reprezentuje pulz o délce *Tb* se shodnou polaritou jakou má pulz předcházející sekvenci nul) (řádek12) v případě je-li *ctnPulz* sudé nebo sekvence ve tvaru X00X (X v tomto případě reprezentuje pulz o délce *Tb* s opačnou polaritou než je pulz předcházejcí této sekvenci nul), je-li ctnPulz liché. V že se generují čtyři nuly v jednom cyklu je uměrně tomu ikrementováno *k*(číslo cyklu) a *ctnPulz*. Je-li počet nul menší než čtyři, je vykonán ELSE (řádek 20) a každá nula se zakóduje zvlášt v nadcházejících cyklích.

```
1 case 'hdb3'
2     ctnPulz = 0;
3     k = 1;
4     while k <= N
5         if Data(k) == 1
6             x = [x A*ones(1,tTb)];
7             A = -A;
8             ctnPulz = ctnPulz+1;
9         elseif k<=N-4 && (Data(k)+Data(k+1)+Data(k+2)+Data(k+3) == 0)
10            if mod(ctnPulz,2) == 0
11                x = [x zeros(1,tTb*3) -A*ones(1,tTb)];
12                ctnPulz = ctnPulz + 1;
13            else
14                x = [x A*ones(1,tTb) zeros(1,tTb*2) A*ones(1,tTb)];
15                ctnPulz = ctnPulz + 2;
16                A = -A;
17            end
18            k = k + 3;
19        else
20            x = [x zeros(1,tTb)];
21        end
22        k = k+1;
23    end
```

Obr. 20 Kód hdb3

2.1.14 DELAY MODULATION

V první smyčce cyklu FOR se vykoná IF na řádku 3, zde se kód rozděluje na další IF-ESLE, kde se vygeneruje sekvence pro logickou jedničku (řádek 5) nebo pro logickou nulu (řádek 8). V první periodě je zavedeno takové pravidlo, že pro logickou jedničku jde o přechod z vysoké úrovně do nízké a pro logickou nulu je po celou periodu vysoká úroveň. V dalších smyčkách cyklu se vykonává pouze ELSE(řádek 10), zde je opět kód rozvětven na několik vnořených if-else příkazů. Pravidlo pro generovaní logické jedničky je zapsáno na řádku 12, pravidlo pro generování logické nuly je pak rozděleno v závislosti zda předchozí hodnota byla logická jedna(řádek 16) nebo nula (řádek 18).

```
1 case 'delay-mod'
2     for k = 1:N
3         if k == 1
4             if Data(k) == 1
5                 x = [x A*ones(1,tTb/2) -A*ones(1,tTb/2)];
6                 A = -A;
7             else
8                 x = [x A*ones(1,tTb)];
9             end
10        else
11            if Data(k) == 1
12                 x = [x A*ones(1,tTb/2) -A*ones(1,tTb/2)];
13                 A = -A;
14            else
15                if Data(k-1) == 1
16                    x = [x A*ones(1,tTb)];
17                else
18                    x = [x -A*ones(1,tTb)];
19                    A = -A;
20                end
21            end
22        end
23    end
```

Obr. 21 Kód delay modulace

2.1.15 2B1Q

Toto kódování má čtyři úrovně pro čtyři různé kombinace dvou bitů. Je-li k-tý bit vektoru Data jednička vykonává se IF na řádku 4, kde se dále rozhoduje vnořeným IF-ELSE zda hodnota k+1 je jednička (řádek 5) nebo nula (řádek 7). V případě že k-tý bit vektoru Data je nula vykoná se ELSE na řádku 10, kde se opět ve vnořeném IF-ESLE na základě hodnoty bitu k+1 rozhodne o úrovni zda se vygeneruje sekvence pro kombinaci 01 (řádek 12) nebo pro kombinaci 00 (řádek 14).

```
case '2b1q'
    k = 1;
    while k <= N
        if Data(k) == 1
            if Data(k+1) == 1
                x = [x -A*ones(1,tTb)];
            else
                x = [x -0.5*A*ones(1,tTb)];
            end
        else
            if Data(k+1) == 1
                x = [x A*ones(1,tTb)];
            else
                x = [x 0.5*A*ones(1,tTb)];
            end
        end
    k = k+2;
    end
```

Obr. 22 Kód 2b1q

2.2 Funkce Upload6022X

Tato funkce upraví data signálu do potřebného formátu a odešle je přes VISA do generátoru Siglent SDG6022X. Funkce má 5 vstupních parametrů jejíž význam je popsán v tabulce 6.

Tab. 6 Parametry funkce UploadSDG6022X

Vstupní parametr	Význam	Datový typ
com	Port, na kterém se naváže spojení	Int
ch	Kanál, na kterém se bude signál generovat	Int
N	Počet bitů vstupních dat	Int
Tb	Bitová perioda	Int
Dat	Data generovaného signálu	Vektor float

Po navázání spojení se na řádku č. 8 pošle dotaz na identifikaci zařízení, odpověď na je poté přečtena z bufferu a vypsána do konzole na řádku 11. Poté se na řádku 12 zajistí aby byl vstupní vektor řádkový a na řádku 13 se vypočte frekvence signálu. Poté následuje úprava dat na požadovaný formát. Generátor Siglent vyžaduje 16-ti bitový formát s dvojkových doplňkem, to znamená, +- 32768. Analogie pro tuto konverzi bude jednodušeji vysvětlena následující tabulkou:

Tab. 7 Formát dat pro SDG6022X

Reprezentace	Amplituda	
	0 až +A	0 až -A
16bit, dvojkový doplněk	0 až 32767	0 až -32768
Čistá dekadická	0 až 32767	65535 až 32768

Nejprve je tedy nutné zjistit amplitudu signálu, tu zjistíme tak, že najdeme maximální absolutní hodnotu, která se nalézá ve vektoru dat signálu (řádek 14). Na následujícím řádku se poté zjistí, jaký bude mít A/D převodník krok pro amplitudu zjištěnou v předchozím kroku. Poté se cyklem **for** vytvoří vektor dat *B* v čistě dekadické reprezentaci vysvětlené tabulkou 6. Pokud je datový bod kladný, vydělí se hodnotou kroku A/D převodníku a zaokrouhlí se k nejbližšímu celému číslu, čím se zajistí že hodnota bude celé číslo a bude se pohybovat mezi 0 a 32767. Pokud by byl datový bod záporný, vydělí se hodnotou kroku A/D převodníku, zaokrouhlí se a navíc se odečte od hodnoty 65535, čímž se zajistí že maximální hodnota bude reprezentována 32768 a minimální 65535. Dále se cyklem **for** na řádku 25 připraví nový vektor, který zakóduje dekadická data vektoru *B* do ASCII reprezentace. To probíhá tak, že se datové body převedou do hexadecimální reprezentace následně se řadou **IF-ELSEIF** doplní nulami na základě délky hex. reprezentace, tak, aby délka byla 2 bajty. Následně se vyčte hexadecimální hodnota spodního (*LB*) a horního (*HB*) bajtu zvlášť a převedou se zpět na dekadickou reprezentaci. Poté se v případě že hodnota datového bodu je nulová, nastaví se oba bajty na hodnot nejbližší nule, kterou lze zapsat, tou je nejnižší hodnota záporné polarity, tzn 65535 (*LB* = 255, *HB* = 255). Je-li pouze jeden bajt nulový, nastaví se nulový bajt do jedničky, neboť VISA má problém akceptovat ASCII charakter 0. A nakonec se do vektoru **z** zapíší ascii znaky reprezentované hodnotami LB a HB. A poté co jsou vytvořeny ascii reprezentace všech bodů se vytvoří hlavička příkazu pro zápis dat do generátoru(řádek 51). Ten je poté odeslán na řádku 53 a posléze se odešle příkaz pro zapnutí požadovaného výstupu a nastaví se generování průběhu.

```
function Upload6022X(app,com,ch,N,Tb,Dat)
    instrreset;
    vu = visa('ni',strcat('USB',num2str(com),'::0xF4EC::0x1101::SDG6XBAX2R0135::0::INSTR'));
    set(vu, 'OutputBufferSize', 1e6);
    set(vu, 'InputBufferSize', 1e6);
    set(vu, 'ByteOrder', 'littleEndian');
    fopen(vu);
    fprintf(vu,'*IDN?');
    outputbuffer = fscanf(vu);
    disp(outputbuffer);
    fprintf(vu, '*RST');
    Data = transpose(Dat);
    f = 1/(N*(Tb/1000));
    Amp = max(abs(Data));
    ADC = Amp/32767;
    B = [];
    for k = 1:length(Data)
        if Data(k) > 0
            B = [B round(Data(k)/ADC)];
        elseif Data(k) <= 0
            B = [B (round(Data(k)/ADC) + 65535)];
        end
    end
    z = [];
    for k = 1:length(B)
         x = B(k);
         x = dec2hex(x);
         if (0 == x)
            x = '0000';
         elseif (1 == length(x))
            x = ['000', x];
         elseif (2 == length(x))
            x = ['00', x];
         elseif (3 == length(x))
            x = ['0', x];
         end
         LB = hex2dec(x(:,5-2:5-1));
         HB = hex2dec(x(:,5-4:5-3));
         if HB == 0 && LB == 0
             HB = 255;
             LB = 255;
         end
         if LB == 0
             LB = 1;
         end
         if HB == 0
             HB = 1;
         end
         z = [z  char(LB) char(HB)];
    end
    SDGstr0 = strcat('C',num2str(ch),':WVDT
WVNM,wave0,FREQ,',num2str(f),',AMPL,',num2str(2*Amp),',OFST,0.0,PHASE,0.0,WAVEDATA,',z);
    fprintf(SDGstr0);
    fprintf(vu, SDGstr0);
    fprintf(vu,strcat('C',num2str(ch),':OUTPUT ON'));
    fprintf(vu,strcat('C',num2str(ch),':ARWV NAME,wave0'));
    fclose(vu);
    delete(vu);
    clear vu;
end
```

Obr. 23 Kód funkce UploadSDG6022X

2.3 Funkce Upload33220A

V případě generátoru Agilent 33220A je uprava dat o něco jednodušší. Počáteční upravy jsou totožné jako u předchozí funkce, nejprve je iniciována komunikace, identifikace zařízení a výpis odpovědi. Poté se vypočte frekvence signálu(řádek 12), vypočte se amplituda signálu(řádek 13) a krok A/D převodníku. V tomto případě jsou data ve 14bitové verzi s dvojkových doplňkem, což odpovídá hodnotám od -8191 až +8191. V případě Agilentu je možné zapsat data přímo v jejich dekadické reprezentaci. Na řádku 17 se opět provádí rozprostření signálu do rozsahu +-8191 a zaokrouhlení k nejbližšímu celému číslu. Poté se hodnota převede na řetězec znaků a je před ni přidána oddělovací čárka(řádek 18). Na konec se hodnota zapíše na konec řetězce **WvStr**, a cyklus se opakuje, dokud se nespracují všechny hodnoty. Po vytvoření řetězce všech hodnot je k řetězci přidána hlavička (řádek 21) a data jsou odeslána do paměti generátoru. A nakonec je nastavena amplituda(řádek 25) a frekvence(řádek 26). Poté se vytvoří kopie průběhu ve volatilní paměti (řádek 27) a je nastaveno generování(řádek 28 a 29). A jako poslední se zapne výstup generátoru na řádku 30 a je ukončena komunikace.

```
1  function Upload33220A(app,com,N,Tb,Data)
2    instrreset;
3    vu = visa('ni',strcat('USB',num2str(com),'::0x0957::0x0407::MY44051396::0::INSTR'));
4    set(vu, 'OutputBufferSize', 1e6);
5    set(vu, 'InputBufferSize', 1e6);
6    set(vu, 'ByteOrder', 'littleEndian');
7    fopen(vu);
8    fprintf(vu,'*IDN?');
9    outputbuffer = fscanf(vu);
10   disp(outputbuffer);
11   fprintf(vu, '*RST');
12   f = 1/(N*(Tb/1000));
13   Amp = max(abs(Data));
14   DAC = Amp/8191;
15   WvStr = [];
16   for i = 1:numel(Data)
17       Data(i) = round(Data(i)/DAC);
18       sChCm = sprintf(',%s', num2str(Data(i)));
19       WvStr = [WvStr sChCm];
20   end
21   arbstring = sprintf('DATA:DAC VOLATILE %s', WvStr);
22   fprintf(vu, arbstring);
23   fprintf(vu, '*WAI');
24   fprintf('Upload succesful...\n\n');
25   fprintf(vu,['VOLT ',  num2str(2*Amp)]);
26   fprintf(vu,['FREQ ',  num2str(f)]);
27   fprintf(vu,'DATA:COPY ARB_1, VOLATILE');
28   fprintf(vu,'FUNC:USER VOLATILE');
29   fprintf(vu,'FUNC:SHAP USER');
30   fprintf(vu,'OUTPUT ON');
31   fclose(vu);
32   delete(vu);
33   clear vu;
34 end
```

Obr. 24 Kód funkce Upload33220A

2.4 Signálové zpracování

2.4.1 Filtrování signálu

K filtrování signálu využívám dvě funkce z toolboxu Communication, které jsou pro potřeby programu ideální. Tyto dvě funkce jsou **rcosine(...)** a **rcosflt(...)**. K filtrování signálu dochází při stisku tlačítka **Filtrovat**,kdy se zavolá funkce události stisku tohoto tlačítka. Nejprve se na základě vybraného typu filtru nastaví proměnná **type**, která se posleze využije v již výše zmíněných funkcích. K tomu docházína řádcích 2 až 9. Dále následuje **IF-ELSE**, kde dochází k filtraci, pokud byla ve výběru filtru zaškrtnuta možnost „Žádný“, vykoná se **IF** a signál se nefiltruje. Je-li vybrána jiná možnost, dojde k filtraci a tudíš je potřeba načíst parametry *Fs/Fd, Roll-off faktor* a *Group delay*, k tomu dochází na řádcích 15 až 17. Na řádku 18 se poté vygenerují kořeny přenosové funkce filtru za pomoci funkce **rcosine(…)** a ty se poté uloží do globálních proměnných k pozdějšímu využití. Následuje načtení vektoru dat z globálních proměnných(řádek 19) a filtrování funkcí **rcosfilt(..)**, zajistí převzorkování i filtraci. Dále je nutné zbavit se vzorku vzniklých group delayem, k tomu dochází na řádku 21. Na řádku 22 se poté vytvoří jemnější vektor času na základě parametru Fs/Fd. A nakonec dojde k zápisu filtrovaného signálu do globálních proměnných.

```
1 function Button9Pushed(app, event)
2   selectedButton = app.ButtonGroup2.SelectedObject;
3   if selectedButton == app.Button_2
4       type = 'none';
5   elseif selectedButton == app.Button2_2
6       type = 'normal';
7   elseif selectedButton == app.Button3_2
8       type = 'sqrt';
9   end
10
11  if strcmp(type,'none')
12      app.v2 = app.v1;
13      app.v2_t = app.v1_t;
14  else
15      FsFd = str2num(app.DropDown3.Value);
16      Roll = str2num(app.DropDown4.Value);
17      GrDel = str2num(app.DropDown5.Value);
18      app.hfil = rcosine(1,FsFd,type,Roll,GrDel);
19      v = app.v1;
20      y2 = rcosflt(v,1,FsFd,type,Roll,GrDel);
21      y2 = y2(GrDel*FsFd:end-GrDel*FsFd);
22      x = (0:length(y2)-1)*1/(FsFd*app.nTb);
23      app.v2 = y2;
24      app.v2_t = x;
25  end
26 end
```

Obr. 25 Filtrování signálu

2.4.2 Výpočet spektra

Spektrální analýza v prostředí matlab je díky integrovaných funkcí velice jednoduchá. Výpočet spektra probíhá v druhé „zobrazovací“ aplikaci stiskem tlačítka, které vykoná následující sérii příkazů. Nejprve dojde k načtení dat průběhu z globálních proměných předaných z hlavní aplikace. Následně dojde na řádku 3 a 4 k výpočtu vzorkovacího kmitočtu na základě časového kroku. Poté se využitím funkce **fft()** získají koeficienty furierovy řady. K výpočtu výkonu dochází na řádku 7 a následně na řádku 8 dochází k interpolaci spektra s faktorem 2. Posléze následuje výpočet vektoru frekvencí na řádku 10. A nakonec dochází k výběru grafu s lineární nebo logaritmickou frekvenční osou.

```
1 x = app.v2_t;
2 y = app.v2;
3 dt = diff(x);
4 fs = (1/mean(dt))*1000;
5 y = fft(y);
6 n = length(y);
7 pow = (abs(y).^2/n)*2;
8 pow2 = interp(pow,2);
9 n2= n*2;
10 freq2 = (0:n2-1)*(fs/n2);
11 ax = app.ButtonGroup.SelectedObject;
12 if ax == app.Button_2
13     plot(app.UIAxes4,freq2,10*log10(pow2/0.001));
14 elseif ax == app.Button2_2
15     semilogx(app.UIAxes4,freq2,10*log10(pow2/0.001));
16 end
```

Obr. 26 Výpočet spektra

2.5 Uživatelské rozhraní

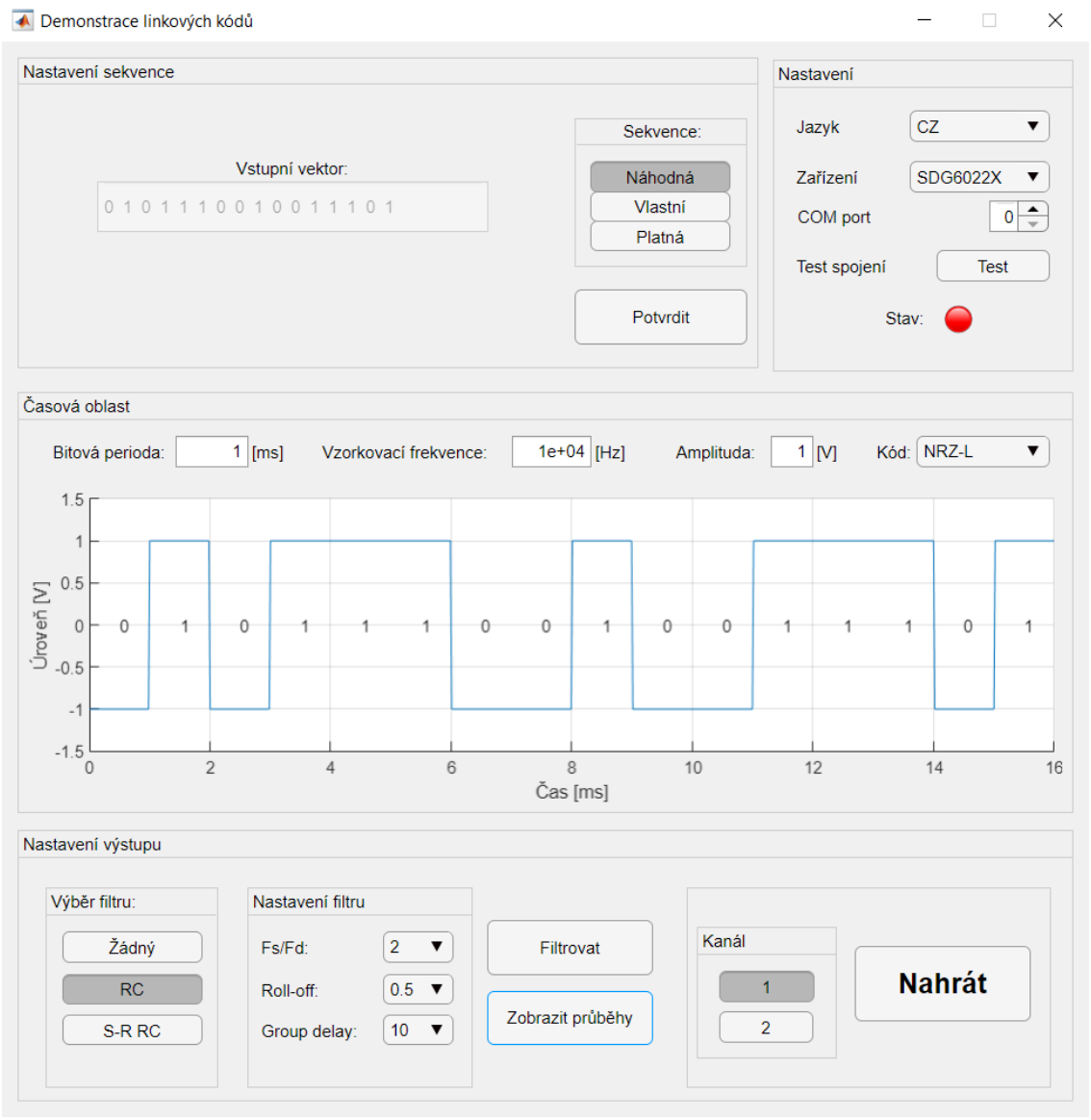
Pro přehlednost uživatelského rozhraní jsem vytvořil dvě aplikace. První aplikace realizuje veškeré kroky od generování signálu přes jeho filtraci až po přenos dat do generátoru. Druhá aplikace je spouštěna tlačítkem **Zobrazit průběhy** z první aplikace a slouží k zobrazování charakteristik filtru a signálu. První aplikace je rozdělena do následujících částí:

- **Nastavení sekvence** - Zde dochází k zadávání a generování průběhů linkových kódů na základě sekvence, amplitudy a bitové periody zadané v části **Časová oblast**. Je zde umožněna volba generování náhodné, vlastní nebo platné sekvence. Tato sekce využívá jedinou funkci a to událost stisknutí tlačítka, ve které se načtou vstupní data z potřebných komponent a následně je zavolána funkce CreateWaveform. Výstupní data jsou uložena do globálních parametrů a průběh signálu je vynesen do grafu v sekci **Časová oblast**.

- **Nastavení** – Zde se nachází volba jazyka celého uživatelského rozhraní, volba zařízení a nechybí zde ani možnost otestovat spojení. Tato část využívá události změny hodnoty DropDown komponenty **Jazyk**, ve které se nastavují veškeré titulky a nadpisy. Ta samá událost je uměle vyvolána ihned po startu aplikace. Dále je zde využito události stisknutí tlačítka, kde se testuje spojení tak, že se naváže komunikace se zařízením a vyšle se dotaz k identifikaci. Pokud se zařízení identifikuje stejným označením jako je označení vybraného zařízení, rozsvítí se kontrolka stavu zeleně. Pokud navázání komunikace selže nebo se zařízení neidentifikuje korektně, dojde k zobrazení varovné hlášky a kontrolka bude červená.

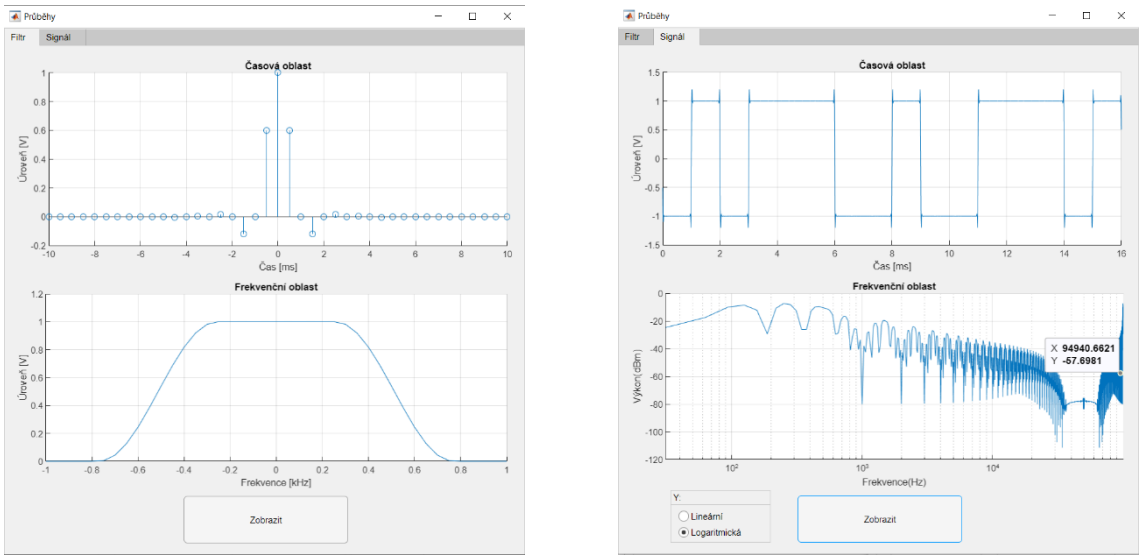
- **Časová oblast –** V této části rozhraní se nachází komponenty pro zadávání parametrů signálu, volby linkového kódu a grafická komponenta pro zobrazení generovaného průběhu. Využívá se zde dvou událostí změny hodnoty komponenty EditField, které jsou využity pro zadávání hodnot bitové periody a vzorkovací frekvence. Tyto dvě komponenty se navzájem ovlivňují, respektive, dojde-li ke změně bitové periody, je vyvolána událost, které na základě bitové periody upraví hodnotu vzorkovacího kmitočtu a naopak, změní-li se hodnota vzorkovacího kmitočtu, dojde ke změně hodnoty bitové periody.

- **Nastavení výstupu –** V neposlední řadě je zde část pro nastavení parametrů výstupu, převzorkování a filtrace signálu. Rozhraní také umožnuje volbu kanálu, na kterém se bude signál generovat. Využívá se zde tří události stisktnutí tlačítka, první z nich je tlačítko **Filtrovat**, to zajišťuje převzorkování původního signálu na nový vyšší vzorkovací kmitočet a jeho následnou filtraci pomocí RC nebo S-R RC filtru s parametry nastavenými v sekci **Nastavení filtru**. Dále se zde využívá události stisku tlačítka **Zobrazit průběhy**, která předá potřebná data druhé aplikaci. A jako poslední je zde událost stisknutí tlačítka **Nahrát**, která na základě vybraného zařízení zavolá příslušnou funkci pro zápis dat do generátoru.



Obr. 27 Vzhled hlavní aplikace

Druhá aplikace zařizuje zobrazený potřebných dat. Aplikace je rozdělena na záložku **Filtr**, kde se zobrazují data vstažená k nastavenému filtru a záložku **Signál**, kde se zobrazuje filtrovaný signál a jeho spektrum.



Obr. 28 Vzhled vedlejší aplikace

3. ZÁVĚR

Cílem práce bylo vytvořit program pro demonstraci linkových kódů, to se mi z velké části povedlo. Nejprve jsem vytvořil funkci pro generování průběhů linkových kódů **CreateWaveform**. Následně jsem vytvořil funkci pro přenos dat do generátoru Siglent SDG6022X **Upload6022X**, jejíž tvorba zabrala nejvíce času. Dokumentace generátoru je velmi stručná a není v ní uvedena jakákoliv speficikace datového formátu pro SCPI příkaz k zápisu dat do generátoru. Ani pátrání po internetu mi nepřineslo úspěch a z toho důvodu funkce není perfektní. V prvním semestru jsem vytvořil funkci, která z počátku přenášela data do generátoru ale ořezávala část signálu. To jsem v druhém semestrů vyřešil a vše fungovala jak mělo. Poté jsem začal řešit uživatelské rozhraní a filtraci signálu. Po filtraci signálu a jeho následném přenosu do generátoru jsem narazil na druhý problém, při jistých kombinacích sekvence, linkového kódu a group delaye dojde k špatnému zápisu dat a signál nemá korektní tvar. Bohužel jsem nebyl schopen přijít na to, jak tento problém vyřešit, při group delay 1 se vždy signál zapíše správně, proto jsem v laboratorním protokolu uvedl měření při group delay 1. Nejprve jsem chtěl zamnknou možnost změny group delay abych zajistil správný zápis v každé situaci, avšak chtěl jsem také umožnit uživateli zobrazit si průběh filtrovaný s různými hodnotami group delaye a tím i vliv group delay na spektrum signálu(v programu). Také jsem měl za úkol navrhnout funkci pro přenos dat do generátoru Agilent 33220A. V tomto případě byl navrh velice jednoduchý díky velmi dobré dokumentaci protokolu pro komunikaci. Funkci jsem nazval **Upload33220A** a funguje bezchybně. Vzhledem k velikosti standalone aplikace jsou v příloze práce odevzdány pouze soubory navržené aplikace, kterou je možné spustit pomocí matlabu nebo pomocí nich vytvořit standalone aplikaci

Literatura

[1] Rábová, Z., Hanáček, P., Peringer, P., Přikryl, P., Křena, B.: Užitečné rady pro psaní odborného textu [online]. Brno: c1993-2008, aktualizováno 2008-11-01 [cit. 2008-11-28]. Dostupné na URL: <http://www.fit.vutbr.cz/info/statnice/psani_textu.html>

[2] Kolektiv autorů: Pravidla českého pravopisu. Academia, Praha, 1993. ISBN 80-200-0475-0

[3] Šablona pro BP/DP a prezentace v2.63 [online]. Brno: FEKT VUT, 2017 [cit. 2017-03-06]. Dostupné z: http://latex.feec.vutbr.cz/sablona/

[4] 2B1Q Signal format. In: *RF Wireless World* [online]. [cit. 2019-12-15]. Dostupné z: https://www.rfwireless-world.com/Tutorials/ISDN-2B1Q-signal-format.html

[5] Ĺine coding. In: *Network Encyclopedia* [online]. [cit. 2019-12-15]. Dostupné z: https://networkencyclopedia.com/line-coding/

[6] LINE CODING IN DIGITAL COMMUNICATION. In: *Fiberoptics4sale* [online]. [cit. 2019-12-15]. Dostupné z: https://www.fiberoptics4sale.com/blogs/archive-posts/95046214-line-coding-in-digital-communication

[7] Line Codes. In: *University of British Columbia: Electrical and Computer engineering* [online]. [cit. 2019-12-15]. Dostupné z: http://www.ece.ubc.ca/~edc/3525.jan2018/lec5.pdf

[8] Line Codes. In: *University of British Columbia: Electrical and Computer engineering* [online]. [cit. 2019-12-15]. Dostupné z: http://www.ece.ubc.ca/~edc/3525.jan2014/lectures/lec7.pdf

[9] Digital Communication - Line Codes. In: *Tutorialspoint* [online]. [cit. 2019-12-15]. Dostupné z: https://www.tutorialspoint.com/digital_communication/digital_communication_line_codes.htm

[10] Line code. In: *Wikipedie* [online]. [cit. 2019-12-15]. Dostupné z: https://en.wikipedia.org/wiki/Line_code

[11] WON LIM, Young. Line Code (3C). In: *Wikimedia Commons* [online]. [cit. 2019-12-15]. Dostupné z: https://upload.wikimedia.org/wikiversity/en/2/2a/DComm.2.C.LineCode.20130920.pdf

[12] Jiří Kozumplík, Radim Kolář, Jiří Jan. ČÍSLICOVÉ ZPRACOVÁNÍ SIGNÁLŮ V PROSTŘEDÍ MATLAB. Vysoké učení technické v Brně, 2001.

[13] Spektrální analýza, doc. Ing. Radim Kolář, PhD. [online]. [cit. 2020-06-01]. Dostupné z: https://moodle-archiv.ro.vutbr.cz/pluginfile.php/379364/mod_folder/content/0/BCZA_Tema13a_spektralni_analyza.pdf?forcedownload=1

Seznam příloh

Příloha č. 1 - Laboratorní protokol

Příloha č. 2 - Zdrojové soubory